- DEFINITIØNS -

une **LISTE** c'est :

(quelquechose$_1$ ... quelquechose$_n$)

abbreviation : qqch ≡ quelquechose

un qqch, ce sera ou bien

un **ATØME** { 1 nombre entier / 1 nom

ou bien

une LISTE.

exemples : atomes :

12
-140
1024
] nombres

BRAVØ
-12-
A21
NIL
] noms

listes : (DØ RE MI FA SØL LA SI)
(DØ MI SØL (FA LA DØ))
((3 5) 7)
(JEAN (PAUL) (PIERRE))
() : liste à 0 éléments.

cette dernière liste, on peut également l'appeler NIL.

- SEPARER ET REGRØUPER -

Il est bon d'avoir la possibilité d'isoler des éléments d'une liste, par ex : isoler le 1er élément. Soit une liste L :

(CAR L) me livre le 1er element de L

(CDR L) me livre L sans son 1er element.

ex : soit L = (A B C)

(CAR L) livre A

(CDR L) livre (B C)

abbreviation "→" ≡ "livre"

On dira que CAR et CDR sont des <u>fonctions</u>
dont la valeur est
pour CAR : le 1er element d'une liste
pour CDR : le reste de la liste.

exemples : L = (DØ MI SØL SI) : 4 elements
(CAR L) => DØ
(CDR L) => (MI SØL SI)

L = ((PAUL) JEAN) : 2 elements
(CAR L) => (PAUL)
(CDR L) => (JEAN)

L = (2 (4 6) 8) : 3 elements
(CAR L) => 2
(CDR L) => ((4 6) 8)

L = (5 JULES 8) : 3 elements
(CAR L) => 5
(CDR L) => (JULES 8)
(CAR (CDR L)) => JULES
(CDR (CDR L)) => (8)
(CAR (CDR (CDR L))) => 8

L = (UN) : 1 element
(CAR L) -> UN
(CDR L) -> ()

je pourrais aussi bien écrire : (CDR L) -> NIL
puisque nous savons que () est la même chose que NIL (par définition).

TERMINØLØGIE : on dira, par ex. pour
(CAR L), que CAR est une fonction
et que L est un argument.
et (CAR L) est un appel de la fonction CAR
qui me ramène en résultat le 1er element de L.

Un appel de fonction doit toujours livrer un résultat, on nomme ce résultat la VALEUR de la fonction.
DONC, CAR et CDR (et leurs combinaisons) nous permettant de séparer des listes en

- leur 1er élément
- le reste de la liste.

EXERCICES :

1 L = ((A) B)

a.) (CAR L) → quoi ?
b.) (CAR (CAR L)) →
c.) (CDR L) →
d.) (CAR (CDR L)) →

2 L = (A (B (C) D) E)

a.) (CDR L) →
b.) (CDR (CDR L)) →
c.) (CAR (CAR (CDR L))) →
d.) (CAR (CDR (CDR L))) →

SOLUTIONS :

1

a.) (A)
b.) A
c.) (B)
d.) B

2

a.) ((B (C) D) E)
b.) (E)
c.) B
d.) E

– REGRØUPER ET SEPARER (suite) –

Voyons à présent comment nous pouvons regrouper des éléments pour former de nouvelles listes.

Soit : X { 1 liste <u>ou</u> 1 atome

Soit : Y 1 liste.

la fonction : (CØNS X Y) livre : la liste Y avec X comme nouveau 1er élément.

<u>ex</u> :
```
X = A
Y = (B)
(CØNS X Y) → (A B)
```

<u>ex</u> :
```
X = A
Y = ()
(CØNS X Y) → (A)
```

<u>ex</u> :
```
X = (A)
Y = (B C)
(CØNS X Y) → ((A) B C)
```

<u>ex</u> :
```
X = DØ
Y = (MI SØL)
(CØNS X Y) → (DØ MI SØL)
```

<u>ex</u> :
```
X = (FA DØ)
Y = ((SØL RE))
(CØNS X Y) → ((FA DØ)(SØL RE))
```

On remarque que, si L est une liste
```
(CØNS (CAR L)(CDR L)) → L
```

EXERCICES : soit X = (A B)
Y = (C)

a.) (CØNS X Y) →
b.) (CAR (CØNS X Y)) →
c.) (CDR (CØNS X Y)) →
d.) (CØNS X (CDR Y)) →
e.) (CØNS (CAR Y) (CDR X)) →
f.) (CDR (CØNS (CAR Y) (CDR X))) →

SØLUTIØNS :

a.) ((A B) C)
b.) (A B)
c.) (C)
d.) ((A B))
e.) (C B)
f.) (B)

Fort heureusement, il existe des abbréviations commodes :

(CADR X) ≡ (CAR (CDR X))
(CADDR X) ≡ (CAR (CDR (CDR X)))

Il est bon de retenir que :

(CAR L) attrape le 1er element de L
(CADR L) attrape le 2d element de L
(CADDR L) attrape le 3ème element de L

exemple : si L = (A (B C) D)
(CAR L) → A
(CADR L) → (B C)
(CADDR L) → D

— LA NØTIØN DE FØNCTIØN —

Nos commandes CAR, CDR, CØNS s'écrivent, comme on vient de le voir :

(CAR qqchose)
(CDR qqchose)
(CØNS qqchose1 qqchose2)

On appelle les qqchoses des <u>arguments</u> et CAR, CDR, CØNS des <u>fonctions</u>. En fait, un objet tel que :

(CAR L)

est un APPEL DE FØNCTIØN, i.e. j'applique la fonction CAR à l'argument L. Cette application doit me livrer un résultat. Dans le cas présent et selon la définition de CAR, c'est le 1er élément de l'argument L.

En général, les appels de fonction devront être écrits selon le schéma suivant :

(nom de fonction arguments1 ... argumentn)

Dans les cas de CAR, CDR et CØNS on dira que
CAR est une fonction à 1 argument.
CDR de même
CØNS est une fonction à 2 arguments.

On notera qu'un appel de fonction est, en lui-même une liste,

<u>ex</u> : si L = (CØNS L1 L2)
(CAR L) → CØNS
(CDR L) → (L1 L2)

et est donc manipulable au même titre que toute autre liste.

– LA NOTION DE PROGRAMME –

IDÉE : quand j'écris l'appel
(CAR L)
je suppose que L est une liste. "L" par lui-même n'est pas une liste, ce n'est qu'un atome. En fait, ce n'est pas de L que je parle, mais d'une liste, par ex. (A B) qui est en quelque sorte ASSOCIÉE à L.
Dans ce cas on dira que (A B) est la VALEUR de L.

BOÎTES : On peut imaginer L comme le nom d'une boîte

L
[ ]

qui "contient" la liste (A B)

L
[(A B)]

PROBLEME : comment placer des listes ou des atomes dans des boîtes ?

exemple : voici une boîte A

A
[ ]

Je veux placer l'atome "14" dans A.
Pour ce faire, j'écrirai : (SETQ A 14)
et du même coup !

A
[14]

IDÉE : l'écriture me permettant cette manipulation sera :
(SETQ nom de boîte objet que je veux placer dans la boîte)

exemple :
3 boîtes A, B et C.

A	B	C
[ ]	[ ]	[ ]

et j'écris :

```
(SETQ A 20)
(SETQ B 4)
(SETQ C -3)
```

ce qui provoque :

A	B	C
[20]	[4]	[-3]

En général, un appel de la fonction SETQ se nommera une AFFECTATION (c'est, et ce n'est que de la terminologie).

Gardons l'exemple précédent. Voici que j'écris :

(SETQ A C)

ATTENTION : le C est le nom d'une boîte. Il est clair que ça ne doit pas faire

A
[C]

mais bien plutôt :

A
[-3]

En effet, si vous vous reportons à la page précédente, la valeur de C (ou le contenu de la boîte C), c'était bien -3.

DONC, quand le nom d'une boîte apparaît en 3ème position d'un appel de SETQ, ça se réfère bien plutôt à la VALEUR de la boîte.

On constatera que j'emploie ici indifféremment les mots de VALEUR et de CONTENU pour une boîte.

Mettons en application ces notions :

exemples. soit 2 boîtes A et B.

A [ ] B [ ]

(SETQ A 3)

A [3] B [ ]

(SETQ B 1)

A [3] B [1]

(SETQ A 2)

A [2] B [1]

(SETQ B A)

A [2] B [2]

(SETQ A B)

A [2] B [2]

(SETQ B 0)

A [2] B [0]

EXERCICES :

[1] 3 boîtes A, B, C.

a.) résultat de l'exécution successive de

```
(SETQ A 5)
(SETQ B A)
(SETQ C B)
```

b.) 2 boîtes DEBUS et BEETHØ

```
(SETQ BEETHØ 1)
(SETQ DEBUS BEETHØ)
(SETQ BEETHØ 2)
```

c.) 3 boîtes B1, B2, B3

```
(SETQ B1 3)
(SETQ B2 -1)
(SETQ B3 B1)
(SETQ B1 B2)
(SETQ B2 B3)
```

SØLUTIØNS :

[1] a.)

A	B	C
5	5	5

b.)

DEBUS	BEETHØ
1	2

c.)

B1	B2	B3
-1	3	3

La notion de PRØGRAMME est déjà largement suggérée par le fait que les appels de SETQ que nous venons de voir sont exécutés <u>à la file</u> (c'est ce qui fait la beauté de la chose). Je pourrai aussi bien dire, c'est la même chose, que les instructions sont exécutées EN SEQUENCE, ou encore séquentiellement, ou encore l'une après l'autre (il est temps pour le lecteur de s'habituer à la langue riche et imprécise des informaticiens).

Voici à présent le classique des classiques :

2 boîtes A [ ] B [ ]

et le problème est : quelque soit leur contenu, on veut les ÉCHANGER. C'est à dire, après l'exécution de quelques SETQ à la file, on veut que l'ancien contenu de A soit dans B, et que l'ancien contenu de B soit dans A.

Essayons, en pensant au transvasement analogue d'un verre de lait et d'un verre d'huile.

```
(SETQ A B)
(SETQ B A)
```

Faisons un peu tourner, avec, par exemple :

A [3] B [10]

L'exécution de (SETQ A B) donne : A [10] B [10]

Du coup, on voit que le "3" est perdu, le lait est dans l'huile, il est clair que ça ne marche pas.

Une idée, c'est d'avoir une 3ème boîte qui servira de réserve. Donnons-nous donc une 3e boîte C et écrivons la séquence :

```
(SETQ C A)
(SETQ A B)
(SETQ B C)
```

et faisons tourner avec :

A [3] B [10] C [ ]

Au départ on a donc :

A [3] B [10] C [ ]

après exécution de (SETQ C A) on a :

A [3] B [10] C [3]

après exécution de (SETQ A B) on a :

A [10] B [10] C [3]

et enfin après exécution de (SETQ B C) on a :

A [10] B [3] C [3]

et la valeur de C, on s'en moque, ce qui compte c'est que les valeurs de A et de B ont été effectivement échangées sans casse.

EXERCICES :

a.) voici 2 boîtes A et B, et 1 boîte C qui contient la liste (HØP GASP)

A [ ] B [ ] C [(HØP GASP)]

Donner les contenus de A, B et C après exécution de :

```
(SETQ A 1)
(SETQ B (CØNS A C))
(SETQ C (CDR C))
```

b.) voici 3 boîtes A1 A2 A3 et 1 boîte L qui contient la liste (DIEU VØUS GARDE)

A1 [ ] A2 [ ] A3 [ ] L [(DIEU VØUS GARDE)]

Écrire une suite d'instructions donnant aux boîtes les valeurs que voici :

A1 [DIEU] A2 [VØUS] A3 [GARDE] L [NIL]

SØLUTIØNS :

a.) A [1] B [(1 HØP GASP)] C [(GASP)]

b.)

```
(SETQ A1 (CAR L))
(SETQ A2 (CADR L))
(SETQ A3 (CADDR L))
(SETQ L NIL)
```

ou encore :

```
(SETQ A1 (CAR L))
(SETQ L (CDR L))
(SETQ A2 (CAR L))
(SETQ L (CDR L))
(SETQ A3 (CAR L))
(SETQ L (CDR L))
```

LA NØTIØN DE QUØTE :

Il est à présent clair que lorsque j'écris, par exemple : (SETQ A B), B est le nom d'une boîte, et ce sera le contenu de la boîte B qui sera placé en A. Mais supposons que je souhaite obtenir : A [B]

c'est bien l'atome B que je veux placer dans A, et pas du tout le contenu de la boîte du même nom. Quand je vais chercher le contenu de la boîte B, on dit que je fais l'EVALUATIØN de B. Pour placer le symbole B dans A, il faut empêcher l'évaluation de B. Pour ce faire, on dispose de l'excellente fonction :

(QUØTE x) → x

exemples : • soit B = 3
(SETQ A B) provoque A [3]
mais :
(SETQ A (QUØTE B)) provoque A [B]

• soit GULP = (A B C)
(SETQ L1 GULP) provoque L1 [(A B C)]
mais :
(SETQ L1 (QUØTE GULP))
provoquera : L1 [GULP]

• soit LØB = (CØNS CAR)
(SETQ LUB LØB) provoque LUB [(CØNS CAR)]
mais (SETQ LUB (QUØTE LØB))
donnera : LUB [LØB]

(SETQ A (CØNS 1 NIL)) donne A [(1)]
mais : (SETQ A (QUØTE (CØNS 1 NIL)))
donne : A [(CØNS 1 NIL)]

- (SETQ S (QUØTE (FA LA DØ)))
  donne : S = (FA LA DØ)

- soit L = (A B)
  (SETQ M (CAR L)) donne : M = A
  mais :
  (SETQ M (QUØTE (CAR L))) donne :
  M = (CAR L)

Il existe une abréviation commode : le caractère '

ex : 'A est équivalent à (QUØTE A)
'(A B) est équivalent à (QUØTE (A B))

exemples :
- (SETQ A 'B) est équiv. à
  (SETQ A (QUØTE B))
- (SETQ L '(SI SØL)) est équivalent à :
  (SETQ L (QUØTE (SI SØL)))

EXERCICES :

[1] à quoi sont équivalents
a.) 'B48
b.) '(CØ2 NACL)
c.) (CAR '(A B))
d.) (SETQ L '(A 'B C))

SØLUTIØNS :

[1].
a.) (QUØTE B48)
b.) (QUØTE (CØ2 NACL))
c.) (CAR (QUØTE (A B)))
d.) (SETQ L (QUØTE (A (QUØTE B) C)))

– LA NOTION DE PROGRAMME (suite) –

Placer des objets dans des boîtes. Cartes [illegible] (ou noms, ou encore variables) doivent [illegible] et ça ne sera pas par miracle.

Faire exister de telles variables (ou boîtes) [illegible] DÉCLARER. "Déclarer" des variables, c'est [illegible] vont exister.

Pour
- se servir de variables
- aligner des instructions, il faut se [illegible]

à l'écriture que voici :

```
( PRØG   liste de     instruction 1 ... instruction n )
         variables
```

Exemple : souvenons nous de l'exemple où nous échangions les valeurs des boîtes A et B.

```
(PRØG   (A  B  C) --|---- je déclare [illegible]   A [ ]  B [ ]  C [ ]
     (SETQ  A  3) --|------------- A [ 3 ]
     (SETQ  B  10) -|------------------ B [   ]
     (SETQ  C  A) --|---------------------- [   ]
     (SETQ  A  B) --|------------- A [ 10 ]
     (SETQ  B  C)) -|------------------ B [   ]
```

6



SOLUTIONS :

[1] a.)

```
(PROG  (A  B  C)
1      2      2
        (SETQ  A  '(DO  RE  MI))
        2          3           32
        (SETQ  B  (CAR  A))
        2          3      32
        (SETQ  C  (CADR  A))
        2          3       32
        (SETQ  A  (CDR  (CDR  A)))
        2          3     4      432
)
1
```

b.)

```
(A  (B  C)  (D
1   2    2  2
        E  (F))  G  H  (I  J))
           3  32        2    21
```

[2] a.) il manque une fermante.
b.) il manque deux fermantes.
c.) HORREUR ! ça commence par une fermante.
d.) tout va bien.

Quand je demande ce qui ne va pas dans des expressions parenthésées, je suppose implicitement que vous savez du même coup reconnaître si une telle expression est correcte.

Savez-vous former toutes les expressions correctes avec

1 paire de parenthèses : ()
2 : (()) , ()()
3 : ((())), (())(), ()(()), (()()), ()()()
4 ?
5 ?

Essayez d'imaginer une méthode parfaitement mécanique pour reconnaître si une expression est correcte ou non.

(Voir pages 97 - 98)

On s'aperçoit déjà qu'on peut se tromper dans les parenthèses. Pour s'y retrouver, je recommande au lecteur (au moment où il écrit ses programmes sur une feuille de papier) l'excellente bricole que voici :

. numéroter les parenthèses par niveau.

exemple :

```
• (A    B)
  1     1

• (A  (B   C))
  1   2     21

• (A (B  C) (E  (F  G)  H  I)  J)   (K)
  1  2    2  2   3    3       2   1   1  1

• (A  (B
  1   2
     (C   D   (E   F)
      3        4    4
        G  (H     (I)     J)  )))
           4      5 5      4  321
```

EXERCICES :

1 numéroter les expressions que voici :

a.)
```
(PRØG   (A  B  C)
        (SETQ  A  '(DØ  RE  MI))
        (SETQ  B  (CAR  A))
        (SETQ  C  (CADR  A))
        (SETQ  A  (CDR (CDR  A)))
      )
```

b.)
```
(A  (B  C) (D
        E  (F))  G  H  (I  J))
```

2 qu'est-ce qui ne va pas dans les expressions que voilà :

a.) `((((((A (B)) C ))) D)`

b.) `(A) (B (C) (D (E F G)`

c.) `) A (B C)`

d.) `(SETQ A (CADDR (CAR A)))`

NOTION D'INITIALISATION :

De même que les boîtes n'existaient pas par miracle, de même, au départ d'un PROG, elles n'auront pas un contenu par miracle. Placer une certaine valeur dans une variable qui n'en a pas encore, c'est effectuer une INITIALISATION (i.e. là où il y avait <u>indéfini</u>, on met <u>quelquechose</u> (liste ou atome, nom ou nombre)).
Dans notre petit programme d'échange, les (SETQ A 3) et (SETQ B 10) étaient des initialisations.

EXERCICE : traduire en LISP le petit projet informel suivant :

- on va se servir des boîtes A, B et C.
- Placer "CIEL" dans C.
- Placer NIL dans A.
- Transférer (copier) la valeur de C dans B.
- Echanger les valeurs de A et de B.

SOLUTION :

```
(PROG (A B C)
  1
      (SETQ C 'CIEL)
      (SETQ A NIL)
      (SETQ B C )
      (SETQ C A )
      (SETQ A B )
      (SETQ B C ))
                1
```

et on a

A	B	C
CIEL	NIL	NIL

J'ai dit tout à l'heure que toute fonction ramène une valeur.

Quelle est la valeur d'un SETQ ? c'est celle du 2d argument.

exemple : (SETQ A 'X) place [A: X] et ramène comme valeur X

exemple : (SETQ A (SETQ B '(DØ SØL)))

A: (DØ SØL) B: (DØ SØL)

exemple : (SETQ A (CAR (SETQ B '(DØ SØL))))

A: DØ B: (DØ SØL)

exemple :

(SETQ A (CAR (SETQ B (CDR '(DØ SØL)))))

A: SØL B: (SØL)

Ça se complique ! Je reconnais que c'est un peu forcé.

- LA NØTIØN DE RETURN :

"RETURN" est une fonction qui permet d'achever, ou de terminer l'exécution d'un PRØG. Elle admet un argument. La valeur de cet argument, ce sera la valeur du PRØG en question.

exemple :

```
(PRØG (A B C)
   (SETQ A 'MØRT)
   (SETQ B '(AU TYRAN))
   (SETQ C (CØNS A B))
   (RETURN C))
```

A: MØRT
B: (AU TYRAN)
C: (MØRT AU TYRAN)

Ici, la valeur du PRØG, c'est la valeur de C, à savoir la liste : (MØRT AU TYRAN).

EXERCICES : quelles sont les valeurs de ces PROG ?

a.)
```
(PROG (M N O)
      (SETQ M 'OUI)
      (SETQ N 'BIEN)
      (SETQ O (CONS N NIL))
      (RETURN (CONS M O)) )
```

b.)
```
(PROG (BLA BLO)
      (SETQ BLA '(IL SUFFIT))
      (SETQ BLO (CONS (CAR BLA) NIL))
      (RETURN (CONS (CADR BLA) BLO)) )
```

SOLUTIONS :

a.) (OUI BIEN)

b.) (SUFFIT IL)

Je rappelle ici que
```
(CONS qqchose NIL)
    donne : → (qqchose)
```
exemples :

- (CONS 'A NIL) → (A)
- (CONS '(A) NIL) → ((A))
- (CONS NIL NIL) → (NIL)

On en conclura que :

(e1 e2 ... en) est une forme abrégée de
(CONS e1 (CONS e2 (... (CONS en NIL) ...)))

exemples :

- (A) est équivalent à : (CONS 'A NIL)
- (A B) " (CONS 'A (CONS 'B NIL))
- (A B C) " (CONS 'A (CONS 'B (CONS 'C NIL)))

Revoir l'exercice ci-dessus à la lumière de ces remarques

- NOTIONS D'ENTREES ET DE SORTIES -

Nous savons initialiser des variables d'un PROG au moyen d'affectations. Ça vous donne des PROG un peu rigides, capables de ne faire qu'une seule chose. On souhaiterai garder le même PROG, mais lui faire traiter d'autres valeurs initiales (sans avoir à réécrire le PROG). Voici comment :

exemple : si j'écris

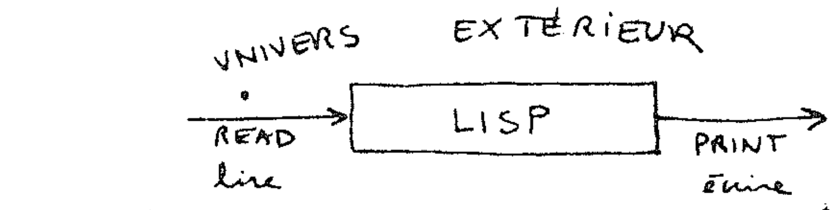
(SETQ A (READ)), ça « lit » une valeur dans l'univers extérieur (à la machine, ou à LISP, comme vous voudrez), et ça place la valeur ainsi lue dans A. Et, supposons que cet univers extérieur a, toute prête à être lue, la liste : (DO RE MI). Du coup ça va placer :

A
| (DO RE MI) |

"READ" est une fonction sans arguments (ou à 0 arguments) qui ramène une valeur qui est lue dans l'univers extérieur (par exemple sur une carte).

abbréviation : UE ≡ Univers Extérieur.

Si on sait lire, on doit pouvoir aussi écrire.



Encore faut-il écrire quelquechose. La fonction qui fait ça c'est : (PRINT x)

ça écrit (par exemple sur une feuille de papier) la valeur de x.

exemples :

- (PRINT 10) : ça écrit "10"

- (PRØG (A)
  (SETQ A -10) (PRINT A))
  : ça écrit "_10"

- (PRØG (A)
  (SETQ A (READ))
  (PRINT A)) : ça lit quelquechose qui se retrouve placé dans la boîte A, puis ça l'imprime.

- (PRØG (A B)
  (SETQ A (READ))
  (SETQ B A)
  (PRINT B)) : ça lit quelquechose dans A, c'est copié dans B, et c'est imprimé.

On peut imaginer qu'il y a à tout instant dans l'UE (univers extérieur) une sorte de queue d'objets qui attendent d'être lus. Imaginez un distributeur automatique, ou encore, un pistolet à chargeur, ou une agrafeuse.

obj
obj
obj
⋮
obj

: les ordres de lecture successifs extraient les objets les uns après les autres.

exemple : A B C D attendent.

lecture ! A B C → D lu.

lecture ! A B → C lu.

etc ....

exemple : voici une queue d'objets prêts à être lus :

```
(DØ RE MI)
(SØL SI)
```

on les extrait à partir d'ici

voici un petit programme :

```
(PRØG (A B C)                    A    B    C
 1
   (SETQ A (READ))               A = (SØL SI)
   (SETQ B (READ))               B = (DØ RE MI)
   (SETQ C 'FA)
   (SETQ B (CØNS C B))           B = (FA DØ RE MI)
   (PRINT B)                     imprime : (FA DØ RE MI)
   (PRINT (CØNS 'MI A))          imprime : (MI SØL SI)
       )
       1
```

REVISIØN : nous avons rencontré les fonctions suivantes :

(CAR l) → 1er élément de l.

(CDR l) → l sans son 1er élément

(CADR l) → 2d élément de l

(CADDR l) → 3e élément de l.

(CØNS x l) → la liste l avec x comme nouveau 1er élément.

(SETQ x y) → place la valeur de y dans la variable x.

(QUØTE x) ou 'x → empêche l'évaluation de x et donne x lui-même.

(PRØG liste de variables ins1 ins2 ... insn) → prépare les variables de la liste, et exécute en séquence les instructions ins1, ..., insn.

(RETURN x) → inclus dans 1 PRØG. Sort du PRØG en ramenant la valeur de x.

(READ) → ramène en valeur un objet lu dans l'UE.

(PRINT x) → imprime dans l'UE la valeur de x.

— UN PEU D'ARITHMETIQUE —

De temps à autre, on est amené à faire du calcul sur des nombres entiers, par exemple trouver le prédécesseur d'un entier, faire une division, etc. Le cas est, en LISP, prévu. Voici quelques fonctions qui agissent sur les entiers, et qui donnent un résultat entier.

Dans ce qui suit, je noterai par x, y, z, x_i, etc..., des valeurs entières.

(ADD1 x) donne $x+1$

(SUB1 x) donne $x-1$

(PLUS x_1 x_2 ... x_n) donne $x_1 + x_2 + \ldots + x_n$ i.e. $\sum_{i=1}^{n} x_i$

(DIFFER x y) donne $x - y$

(TIMES x_1 x_2 ... x_n) donne $x_1 * x_2 * \ldots * x_n$ i.e. $\prod_{i=1}^{n} x_i$

Nous, programmeurs, notons "*" l'opérateur de multiplication.

(QUO x y) donne x / y.

Nous notons également par "/" (prononcez SLASH) l'opérateur de division.

EXEMPLES : Soit X = 6, Y = 5, Z = −3.

- (ADD1 1) → 2
- (ADD1 2) → 3
- (ADD1 0) → 1
- (ADD1 −5) → −4
- (ADD1 X) → 7
- (ADD1 Z) → −2
- (SUB1 0) → −1
- (SUB1 Y) → 4
- (SUB1 Z) → −4
- (ADD1 (SUB1 1)) → 1

```
. (PLUS 1 2 3) → 6
. (PLUS 1 5 -1) → 5
. (PLUS Y 7) → 12
. (PLUS 2 (ADD1 X) 9 (SUB1 Z)) → 14
. (DIFFER 0 1) → -1
. (DIFFER X Y) → 1
. (TIMES 1 2 3 4) → 24
. (TIMES (TIMES 2 Y) Z) → -30
. (QUØ 4 2) → 2
. (QUØ 10 5) → 2
. (QUØ 18 3) → 6
. (QUØ 3 3) → 1
. (QUØ 3 6) → 0
              ↑ bizarre n'est-ce pas ?
. (QUØ 5 2) → 2
              ↑ de plus en plus bizarre !
```

Voilà l'idée :

normalement 7 /2 = 3.5

(le "." chez nous remplace avantageusement la virgule).

Enlevez le point et ce qui suit, ça fait : 7 /2 = 3

. Donc, enlever du résultat d'une division le point et ce qui le suit définit ce qu'on appelle une DIVISION ENTIÈRE.

Entraînons-nous un peu :

```
(QUØ 4 2) → 2
(QUØ 5 2) → 2     car 2(.5) éliminés.
(QUØ 6 2) → 3
(QUØ 7 2) → 3
(QUØ 2 3) → 0
```

L'idée est : quand la division tombe juste, on l'emploie comme d'habitude ; si elle ne tombe pas juste, on garde la PARTIE ENTIÈRE du résultat

ex : 3 . 1416 (3 : partie entière ; 1416 : partie décimale)

Il existe également la fonction

(REM x y) → reste de la division de x par y.

Cette dernière fonction donne lieu à une foultitude d'usages épatants que nous verrons bientôt.

EXERCICES :

1.) (ADD1 (SUB1 0)) → ?
2.) (SUB1 (ADD1 0)) →
3.) quelque soit x
(ADD1 (SUB1 x)) →
4.) (PLUS 1 2 3 4 5 6 7) →
5.) (TIMES 1 2 3 4 5 6 7) →
6.) (ADD1 (ADD1 (ADD1 0))) →
7.) (QUØ 100 50) →
8.) (QUØ -100 50) →
9.) (QUØ 19 5) →
10.) (QUØ 3 4) →
11.) (QUØ 19 45) →
12.) (REM 10 2) →
13.) (REM 9 2) →
14.) (QUØ (PLUS 5 5) (TIMES 1 2)) →
15.) (DIFFER 8 5) →
16.) (DIFFER 19 20) →
17.) (DIFFER x x) →

SØLUTIØNS :

1.) 0
2.) 0
3.) x
4.) 28
5.) 5040
6.) 3
7.) 2
8.) -2
9.) 3
10.) 0
11.) 0
12.) 0
13.) 1
14.) 5
15.) 3
16.) -1
17.) 0

— LA NOTION DE TEST —

INTRODUCTION : Au point où nous en sommes, nous sommes déjà en mesure de faire d'excellents trucs.

exemple : avaler une liste, et faire imprimer ses éléments les uns après les autres.

voici la queue :

| (DO RE MI) |

et le programme.

```
                                        lignes
(PROG   (L)
        (SETQ  L  (READ))                 1
        (PRINT  (CAR  L))                 2
        (SETQ  L   (CDR  L))              3
        (PRINT  (CAR   L))                4
        (SETQ  L  (CDR   L))              5
        (PRINT   (CAR   L))               6
        (SETQ  L   (CDR   L))             7
        (RETURN   'BRAVO)  )              8
```

si j'inscris dans une colonne les valeurs successives que prend L au cours du programme, j'ai :

L	en
(DO RE MI)	1
imprime : DO	2
(RE MI)	3
imprime RE	4
(MI)	5
imprime MI	6
()	7
imprime BRAVO	8

La liste n'avait ici que 3 éléments. Si elle avait eu 20, 100 ou 1000 éléments, il aurait fallu un autre programme affreusement long. Situation impossible ! Essayons d'imaginer quelque chose de plus économique.

Ça pourrait être quelque chose comme ça :

"imprimer les CAR de la liste tant qu'elle n'est pas vide"

ou bien encore (c'est la même chose)

"regarder si la liste est vide ; si elle ne l'est pas, imprimer son CAR ; puis réduire la liste à son CDR (en quelque sorte, "avancer" dans la liste), et recommencer le tout"

ou bien encore (faites bien attention !)

```
boucle :  tester si la liste est vide.
          si elle est vide arrêter.
          sinon . imprimer le CAR de la liste
                . remplacer la liste par son CDR
                . retourner à "boucle".
```

Prenons les problèmes un par un.

Tout d'abord : "tester si une liste est vide".

Il existe une fonction qui fait ça, c'est :

(NULL l), qui ramène comme valeur :

- T si l est vide (i.e. est () ou NIL)
- NIL si l n'est pas vide

"T" c'est un atome comme un autre. Il est simplement différent de NIL. Par définition, la valeur de T, c'est T lui-même.

exemples :

- (NULL NIL) → T
- (NULL ()) → T
- (NULL '(A B)) → NIL
- (NULL T) → NIL
- (NULL 'A) → NIL
- (PROG (A)
  (SETQ A NIL) (PRINT (NULL A)))
  imprime T
- (NULL (NULL T)) = T
- (NULL (NULL NIL)) = NIL

```
• (PROG (A) (SETQ A (READ))
            (PRINT (NULL A))   ----> imprime NIL
            (SETQ A (CDR A))
            (PRINT (NULL A))   ----> imprime T
  )
```

avec la queue

| (A) |

EXERCICES : donner la valeur de

[1]

a.) (NULL (NULL (NULL T)))

b.) (NULL (NULL (NULL ())))

c.) sachant que L = (A (B))

c.1) (NULL (CAR L))

c.2) (NULL (CDR (CDR L)))

c.3) (NULL (CADR L))

c.4) (NULL (CDR (CADR L)))

[2] sachant que L = (A NIL A)

a.) (CAR L) → ?

b.) (CADR L) →

c.) (NULL (CADR L)) →

[3] sachant que L = (() (()) A)

a.) écrire L d'une autre façon

b.) (CADR L) →

c.) (NULL (CADR L)) →

d.) (NULL (CAR (CADR L))) →

SOLUTIONS :

[1]

a.) NIL

b.) T

c.1) NIL

c.2) NIL

c.3) NIL

c.4) T

[2]

a.) A

b.) NIL

c.) T

[3]

a.) (NIL (NIL) A)

b.) (NIL)

c.) NIL . en effet "(NIL)" comporte 1 élément, et n'est donc pas vide.

d.) T

— LA NØTIØN DE CØND —

Elle permet d'exploiter le résultat obtenu à la suite d'une question, par exemple :

(NULL qqchose)

si réponse T faire action1

si réponse NIL faire action2

La fonction CØND devra être écrite suivant le schéma :

(CØND couple1 couple2 ... couplen)

chacun des couples sera : (question conséquence)

exemple : si A = NIL alors placer 3 dans B, si A ≠ NIL placer 6 dans B

```
(CØND ((NULL A) (SETQ B 3))      ——> couple1
        question   conséquence
couple2 <—— (T (SETQ B 6))    )
            question conséquence
             └> qui ramène toujours la valeur T.
```

Je rappelle que, par définition, la valeur de T c'est T.

EVALUATIØN DU CØND : on avance de gauche —> droite, couple par couple, et, à la 1ère question qui donne une réponse différente de NIL, on exécute la conséquence correspondante, et on achève ainsi l'évaluation du CØND.

exemples : " si A ≠ NIL alors placer NIL dans B sinon placer NIL dans C "

```
(CØND
    ((NULL A) (SETQ C NIL))
    (T (SETQ B NIL))  )
```

qui peut également s'écrire :

```
(CØND
    (A (SETQ B NIL))
    (T (SETQ C NIL))  )
```

ici " (A ... veut dire : valeur de A, ou encore, contenu de la boîte A.

T est le nom d'une boîte standard qui contient par définition T.

exemple : "si A = NIL alors placer 3 dans B, sinon si B = NIL alors placer -1 dans B"

```
(CØND   ((NULL A) (SETQ B 3))
        ((NULL B) (SETQ B -1)) )
```

EXERCICES :

écrire les CØND correspondants aux énoncés suivants

a.) "si B1 ≠ NIL alors placer 1 dans A"

b.) "si B1 = NIL alors placer 1 dans A sinon placer 2 dans A.

c.) "si B1 ≠ NIL alors placer (CDR L) dans L sinon si B2 = NIL alors placer (CAR L) dans L sinon placer (CADR L) dans L.

SØLUTIØNS :

a.)
```
(CØND (B1 (SETQ A 1)) )
```

b.)
```
(CØND
    ((NULL B1) (SETQ A 1))
    (T (SETQ A 2)) )
```

ou encore :

```
(CØND
    (B1 (SETQ A 2))
    (T (SETQ A 1)) )
```

ou encore :

```
(SETQ A (CØND
           (B1 2)
           (T 1)) )
```

On remarquera que la valeur d'1 CØND est celle de la conséquence exécutée.

c.)
```
(CØND
    (B1 (SETQ L (CDR L)))
    ((NULL B2) (SETQ L (CAR L)))
    (T (SETQ L (CADR L))) )
```

ou encore :

```
(SETQ L (CØND (B1 (CDR L))
              (B2 (CADR L))
              (T (CAR L)) ))
```

si toutes les questions d'1 CØND donnent la valeur NIL, la valeur du CØND est dans ce cas [illegible].

- LA NØTIØN D'ETIQUETTE -

Dans un PRØG, les instructions sont normalement exécutées les unes après les autres. Il y a moyen cependant de RØMPRE cette succession, et de revenir exécuter des instructions sur lesquelles on est déjà passés ;

exemple :

```
(PRØG (X Y)
        instruc1   ↓
        instruc2
        instruc3   ↓
        instruc4   ↓
           ⋮       ↓
```

et on aimerait réexécuter les instructions 2, 3, 4, etc. L'idée c'est d'écrire le PRØG en plaçant devant l'instruction 2 un atome, qui, du coup, NØMME la suite d'instructions 2, 3, ...

exemple :

```
(PRØG (X Y)
        instruc1
   HØP  instruc2
        instruc3
        instruc4
           ⋮
```

Mais, dans ce cas, après l'instruc4, il faut trouver un moyen de "retourner" à l'instruction "étiquetée" par HØP. L'atome HØP devient ici l'ETIQUETTE de l'instruc2.

Il existe une fonction (GØ etiquette) qui, quand elle est exécutée, provoque la reprise de la suite d'instructions qui suivent l'étiquette.

Voici un exemple d'étiquette.

Je veux lire une liste, que je suppose non vide, et imprimer séparément tous ses éléments, puis arrêter. La liste pourra bien être de longueur quelconque.

```
(PRØG   (L)
1
         (SETQ   L  (READ))
ENCØRE   (PRINT  (CAR  L))
         (SETQ   L  (CDR   L))
         (CØND
             (L  (GØ   ENCØRE)))
         (RETURN   'BRAVØ) )
                          1
```

ICI : "ENCØRE" est une étiquette.

FAISØNS TØURNER :

. on a, à la queue, une liste, par exemple qui attend d'être lue. | (A B C) |

→ on exécute (SETQ L (READ)) ce qui provoque
L: | (A B C) |

→ on ignore l'étiquette en passant dessus.

→ on exécute (PRINT (CAR L)), ce qui imprime A.

→ on exécute (SETQ L (CDR L)) ce qui place
L: | (B C) |

→ on teste si L = NIL. Comme ce n'est pas le cas, on "retourne" à l'étiquette "ENCØRE".

→ on exécute (PRINT (CAR L)) ce qui imprime B.

→ on exécute (SETQ L (CDR L)) ce qui place
L: | (C) |.

→ on teste si L = NIL. Comme ce n'est pas le cas, on "retourne" à l'étiquette "ENCØRE".

→ on exécute (PRINT (CAR L)) ce qui imprime C.

→ on exécute (SETQ L (CDR L)) qui place
L: | NIL |

→ on teste si L = NIL. Comme c'est le cas, on NE RETØURNE PAS à "ENCØRE" mais on va à la suite.

→ on sort du PRØG avec comme valeur "BRAVØ".

→ fini.

- TESTS ARITHMETIQUES -

Il y a des moments, dans la vie, où on est content de pouvoir savoir si un entier est plus grand, plus petit, ou égal à un autre, ou encore si un entier est égal à zéro ou non. Ce sont donc des questions auxquelles on répond par OUI ou par NON, selon la correspondance :

OUI ⟷ T

NON ⟷ NIL

Voici donc des fonctions de <u>comparaison</u> entre entiers

(LT x y) → { T si $x < y$; NIL si $x \geq y$ }

(GT x y) → { T si $x > y$; NIL si $x \leq y$ }

(EQ x y) → { T si $x = y$; NIL si $x \neq y$ }

(ZEROP x) → { T si $x = 0$; NIL si $x \neq 0$ }

exemples :

```
(LT 3 5) → T
(LT -3 0) → T
(LT 3 2) → NIL
(LT -4 -5) → NIL
(GT 8 1) → T
(GT -1 -1048) → T
(GT 1 1) → NIL
(GT 9 1048) → NIL
(EQ 3 3) → T
(EQ -1 -1) → T
(EQ 1 -1) → NIL
(ZEROP 0) → T
(ZEROP 4095) → NIL
```

REVISION : soit A [1] HOP [8]

```
(PLUS A HOP) → 9
(ADD1 HOP) → 9
(PLUS A HOP 1) → 10
(TIMES (ADD1 A) HOP -1) → -16
(QUO HOP (ADD1 A)) → 4
(DIFFER HOP A) → 7
(DIFFER A HOP) → -7
```

EXERCICES DE REVISIØN : soit A = -5, B = 3, C = 7

a.) (PLUS A B C) → ?
b.) (TIMES C C) →
c.) (TIMES B B B) →
d.) (DIFFER C B) →
e.) (DIFFER C A) →
f.) (DIFFER A B) →
g.) (TIMES A A A) →
h.) (ADD1 (DIFFER B B)) →
i.) (QUØ C B) →
j.) (QUØ (PLUS 3 C) (SUB1 B)) →
k.) (REM C 2) →
l.) (REM C B) →

SØLUTIØNS :

a.)	5	g.)	-125
b.)	49	h.)	1
c.)	27	i.)	2
d.)	4	j.)	5
e.)	12	k.)	1
f.)	-8	l.)	1

A présent, essayez de tirer une conclusion générale de l'observation de ce qui suit :

(REM 0 2) → 0
(REM 1 2) → 1
(REM 2 2) → 0
(REM 3 2) → 1
(REM 4 2) → 0
(REM 5 2) → 1
(REM 6 2) → 0
(REM 7 2) → 1
(REM 8 2) → 0
(REM 9 2) → 1
(REM 10 2) → 0
(REM 11 2) → 1

Qu'en pensez vous ?

EXERCICES : voici A [1] B [3] C [9]

[1]

a.) (LT (ADD1 A) (SUB1 (SUB1 B))) ...)

b.) (ZERØP (PLUS A C (DIFFER [illegible] 10)))

c.) (GT A (DIFFER B 2))

[2] voici A [(1 2)] B [4] C [(-5)]

a.) (PLUS (CAR A) B)

b.) (PLUS (ADD1 B) (CAR C))

c.) (GT (CAR A) (CADR A))

d.) (LT (CADR A) (CAR A))

e.) (ZERØP (PLUS B -4))

f.) (EQ (ADD1 B) (CAR C))

g.) (DIFFER (CAR A) (ADD1 B))

h.) (TIMES (CADR A) B (CAR C))

SØLUTIØNS :

[1] a.) c'est équivalent à (LT 2 1) → NIL

b.) T

c.) c'est équivalent à (GT 1 1) → NIL

[2] a.) c'est équivalent à (PLUS 1 4) → 5

b.) 0

c.) NIL

d.) NIL

e.) T

f.) T

g.) -10

h.) -40

- CØNSTRUCTIØN CØMMENTÉE DE PRØGRAMMES -

Nous allons . avaler une liste
. compter combien elle comporte d'éléments
. imprimer le nombre en question

```
1 ------          (PRØG (L K)
2 ------               (SETQ K 0) (SETQ L (READ))
3 -----   HØP          (CØND
4     -----               ((NULL L)(GØ ØK))
5  -----               (SETQ K (ADD1 K))
6  -----               (SETQ L (CDR L))
7  ------              (GØ HØP)
8  ----   ØK           (PRINT K)
9      -----           (RETURN 'BRAVØ) )
```

J'ai numéroté les lignes pour que les explications soient plus claires.

LIGNE 1 : je réserve 2 boîtes [L] et [K].
Pourquoi ? L contiendra la liste que je vais lire.
K contiendra le nombre d'éléments que je cherche. Dans notre jargon, on dira que K est un <u>compteur</u>. Etre un compteur, c'est en effet servir servir à CØMPTER des trucs, ici, combien il y aura d'éléments dans L.
Voilà pourquoi j'ai réservé ces 2 boîtes.

LIGNE 2 : (SETQ K 0) . Je place [K: 0]. Pourquoi ?
Parceque, au fur et à mesure que je vais trouver des éléments dans la liste L, je vais ajouter 1 à K.
Imaginez la situation que voici :
[K: vide], et je fais (ADD1 K). Ça paraît assez étrange d'ajouter 1 à RIEN, car RIEN n'est pas un nombre. Tandis que si j'ai, au départ [K: 0], alors, à la bonne heure, je sais bien que (ADD1 K) = 1, car 0+1 = 1. Vu ?

Voilà pourquoi je met 0 dans K.
Donc (SETQ K 0) INITIALISE K à 0.

LIGNE 2 (SUITE) : (SETQ L (READ)) avale tout simplement une liste lue dans l'univers extérieur, et la place dans la boîte L.

exemple :

QUEUE
| (DØ RE MI) |

et je fais (SETQ L (READ)), ça provoque

L
| (DØ RE MI) |

Nous voyons bien que la liste comporte 3 éléments, mais le programme ne le sait pas encore.

LIGNE 3 : Je passe gaiement sur l'étiquette HØP sans m'en apercevoir, et je tombe sur un CØND. On va donc tester quelque chose, quoi ? vous le saurons à la

LIGNE 4 : ça teste si L contient quelquechose ou non. Autrement dit, si L = NIL, on s'ordonne d'aller executer les instructions qui suivent l'étiquette ØK (ligne 8), où vous remarquez qu'on imprime alors le contenu de la boîte-compteur K, puis on quitte le programme, grâce au RETURN qui ramène un BRAVØ bien mérité.

REMARQUE : Vous voyez que si, ligne 2, j'avais eu

queue
| () |

et (SETQ L (READ)), ça aurait placé NIL dans L

L
| NIL |.

Du coup, en ligne 3-4, on aurait sauté directement à ØK où ça aurait imprimé zéro. C'est donc cohérent avec l'idée que NIL, ou (), c'est une liste à 0 éléments.

LIGNE 5 : (SETQ K (ADD1 K)). Si on a pu arriver là, c'est qu'il y avait encore des éléments dans L. Au moins 1 élément. On augmente de 1 notre petit compteur K.

LIGNE 6 : (SETQ L (CDR L)). On avance d'un cran dans la liste L.

- exemple :

AVANT L = (DØ)
(SETQ L (CDR L))
APRES L = ()

- exemple :

AVANT L = (DØ RE)
(SETQ L (CDR L))
APRES L = (RE)

- exemple

AVANT L = (DØ RE MI)
(SETQ L (CDR L))
APRES L = (RE MI)

Vous voyez qu'avec cette façon de faire, on épuise un à un les éléments de L, et qu'il arrivera bien un moment où elle se retrouvera avec L = ()
(Pensez au petit test de la ligne 3-4).

LIGNE 7 : Elle vous envoie aux instructions qui suivent immédiatement l'étiquette HØP. C'est justement le test de L vide ou pas ; et on s'y renvoie avec un élément de moins dans L (grâce à la ligne 6). Donc si il n'y avait plus qu'1 élément dans L, on se retrouve à HØP avec 0 éléments, et on se retrouve lancé à ØK, où on imprime le nombre K.

LIGNE 8 - 9 : on l'a déjà vu, quand le programme est presque fini, on imprime le contenu de K, et on sort du programme avec la valeur BRAVØ.

STRUCTURE GLØBALE DU PRØGRAMME :

- . initialiser K à zéro.
- . lire dans L

HØP . si L = NIL vers FIN
- . ajouter 1 à K
- . enlever l'élément de tête de L (donc il y a un élément en moins)
- . VERS HØP

FIN . imprimer K
- . arrêter, avec la valeur BRAVØ

On va à présent faire un peu tourner l'objet à la main (faire tourner à la main, c'est ce que ferait l'ordinateur ou LISP, s'il était là, avec ce programme). Attention, ça commence :

<u>ligne 1</u> L [ ] K [ ]

<u>ligne 2</u> queue [(DØ RE)] AVANT

queue [ ] APRES K [0] [(DØ RE)]

<u>ligne 3-4</u> L ≠ NIL, donc on ne va pas à FIN, on continue à la suite.

<u>ligne 5</u> K [1]

<u>ligne 6</u> L [(RE)]

<u>ligne 7</u> vers HØP

<u>ligne 3-4 :</u> L ≠ NIL donc on ne va pas à ØK, et continue à la suite

<u>ligne 5 :</u> K [2]

<u>ligne 6 :</u> L [()]

<u>ligne 7 :</u> VAS HØP

<u>ligne 3-4 :</u> Attention, à présent il y a bien NIL dans L. Donc on va à ØK.

<u>ligne 8 :</u> on imprime le contenu de K, à savoir <u>2</u>.

<u>ligne 9 :</u> On sort du PRØG avec la valeur BRAVØ. Bravø !

A présent relisez tout çà calmement.

Puis, faites tourner vous même à la main, avec cette liste dans la queue.

queue [(BØN)]

Ne cherchez pas vraiment à comprendre cette autre façon de calculer la longueur d'une liste, mais regardez-là tout de même.

```
(DE LENGTH (L)
    (CØND
        ((NULL L) 0)
        (T (ADD1 (LENGTH (CDR L)))) ))
```

• Voir page 85

. ENCØRE UN PRØGRAMME [illegible]

Ce coup-ci, on [illegible] chercher combien de fois il y a [illegible] par exemple l'atome FA.

exemple : dans (3 9 A DUR FA HØP)
il y a 1 fois l'atome FA.

exemple : dans (O FA A FA FA B), il y a
3 fois l'atome FA.

exemple : dans (DØ MI SØL SI), il y a 0 fois
l'atome FA.

Essayons de programmer cet objet-là.
Je réutilise la numérotation des lignes (mais vous, si vous tapez ça sur cartes, ne tapez surtout pas les numéros de ligne).

```
1 ----  (PRØG (K L)
2 ----       (SETQ K 0)
3 ----       (SETQ L (READ))
4 ---  HØP   (CØND
5 ---          ((NULL L) (GØ ØK))
6 ----         ((EQ (CAR L) 'FA)
7 ---            (SETQ K (ADD1 K))))
8 ----       (SETQ L (CDR L))
9 ---        (GØ HØP)
10 -   ØK    (RETURN K) )
```

Vous voyez, ça ressemble assez à la structure du programme de la page 36. On réserve aussi un compteur K et une boîte L pour la liste lue.

Un petit mot de terminologie. Ici, [illegible] combien de fois un [illegible] [illegible] liste L. Une [illegible] façon [illegible]

c'est à dire qu'on compte les ØCCURENCES de l'élément dans la liste.

exemple : (RE MI LA SI RE MI LA SI RE)

l'atome SI a 2 occurences dans la liste. Vu ?

Revenons à votre programme, au lieu de le disséquer, je le fais ici directement tourner à la main. C'est parti.

ligne 1 — K [ ] L [ ]

ligne 2 — K [0]

ligne 3 — quote, par exemple [(DØ FA RE)] — L [(DØ FA RE)]

ligne 4-5 — on regarde si L = NIL, si øui, on va à ØK. Ici, ce n'est pas le cas.

ligne 6-7 — on regarde si (CAR L) est égal à FA. Si c'est oui on augmente le compteur K de 1 (on dit qu'on INCRÉMENTE K). Si c'est non, on va à la suite.

remarque : EQ (voir p. 33) marche aussi pour les atomes non-numériques.

Ici, (CAR L), c'est DØ, donc on laisse K dans son état, et on va à la suite.

ligne 8 — on avance dans L. — L [(FA RE)]

ligne 9 — vers HØP.

ligne 4-5 — est-ce que L = NIL ? NØN. On va à la suite.

ligne 6-7 — est-ce que (CAR L) = FA ! øui

donc K [1]

ligne 8 — on avance dans L. — L [(RE)]

ligne 9 — vers HØP

ligne 4-5 — est-ce que L = NIL ? NØN. A la suite.

ligne 6-7 — est-ce que (CAR L) = FA ? NØN. On ne touche donc pas à K.

ligne 8 — L [()] i.e. NIL

ligne 9 vers HØP

ligne 4-5 est-ce que L = NIL ? Parfaitement !

On va donc à l'étiquette ØK.

ligne 10 on sort du PRØG, en ramenant sous son bras la valeur (le contenu) de K, à savoir 1.

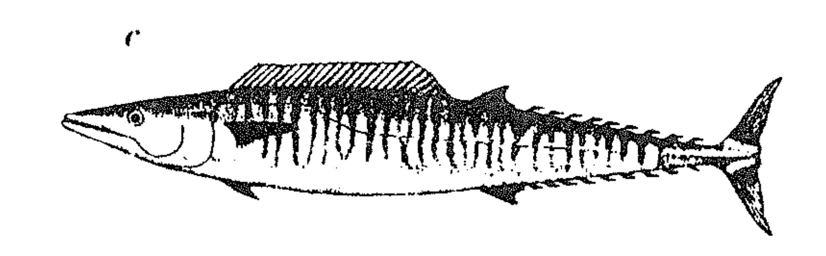
- fini -

Relisez tout ça tranquillement.

Allez donc taper ce bedit brogramme sur cartes, et faites le tourner avec un VRAI ordinateur, et différentes listes lues. Ça fait le plus grand bien, ça fixe les idées, et ça donne le coup d'oeil pour compter les parenthèses.

De plus, jetez un coup d'oeil, sans vraiment approfondir, sur cette autre manière de compter combien une liste comporte de FA.

```
(DE   FA-CØMBIEN   (L ;; K)
1
      (SETQ  K  0)
      (WHILE  L
      2
         (AND  (EQ  (NEXTL  L)  'FA)
         3      4                    4
               (SETQ  K  (ADD1  K)) ))
                4                  4  32
      K)
       1
```



EXERCICE : Qu'est-ce qui ne vas pas dans cette ligne isolée de programme ?

ENCØRE (GØ ENCØRE)

SØLUTIØN : imaginez un serpent qui se mord la queue, ou encore ce petit dessin :

On dit, dans notre jargon (qui est aussi le vôtre) que le programme BØUCLE. La pauvre bête exécute (GØ ENCØRE), ça la renvoie exactement au même endroit, où elle recommence ad infinitum.

Quand ça arrive (ça arrive assez souvent) faut achever l'animal au fusil à éléphant.

On peut généraliser cette affreuse situation : voyez plutôt.

```
RE      (GØ ENCØRE)
  ⋮
ENCØRE  (GØ RE)
```

C'est encore le même désastre.

EXERCICE : écrire un programme qui

- lit un objet ; si c'est NIL terminé.
- sinon lit un 2d objet.
- imprime la somme des 2 objets (ce sont bien évidemment des nombres).
- recommence le tout.

SØLUTIØN :

```
(PRØG   (A   B)
1
   RE   (SETQ  A  (READ))
        (CØND
           ((NULL  A) (RETURN  'FINI)))
        (SETQ  B  (READ))
        (PRINT  (PLUS  A  B))
        (GØ  RE)   )
                   1
```

Dans ce programme, avec, par exemple, la queue que voici :

```
NIL
7
3
5
-2
4
21
```

matérialisée par la carte-donnée suivante :

```
21   4   -2   5   3   7   ()
```

seront successivement imprimés :

```
25
3
10
FINI
```

EXERCICE : écrire un programme qui

- lit une liste ; si c'est NIL terminé. sinon, c'est une liste de nombres
- imprime après recherche, le plus grand nombre de la liste. (le MAXimum).
- recommence le tout.

SØLUTIØN :

```
(PRØG  (MAX  L)
   RE       (SETQ  L  (READ))
            (CØND
               ((NULL  L) (RETURN  'ØUF )))
            (SETQ  MAX  (CAR  L ))
   ENCØRE   (SETQ  L  (CDR  L ))
            (CØND
               ((NULL  L) (PRINT  MAX)
                          (GØ  RE ))
               ((GT  (CAR  L) MAX)
                  (SETQ  MAX  (CAR  L )) ))
            (GØ  ENCØRE)
)
```

Dans ce programme, avec, par exemple, cette queue

```
()
(-1  7  3)
(9  4)
(1  3  1024  6)
(3)
```

matérialisée par la carte-données suivante :

```
(3)  (1  3  1024  6)  (9  4) (-1  7  3)  ()
```

seront successivement imprimés :

```
3
1024
9
7
ØUF
```

Etudiez <u>très</u> soigneusement ce programme, et faites le tourner à la main, et à l'ordinateur.

REMARQUE : vous constatez qu'après le ENCØRE, le CØND a la structure :

$$\underset{2}{(}\text{CØND } \underset{3}{(}(\text{NULL L})(\text{PRINT MAX})(\text{GØ RE})\underset{3}{)} \ \ldots \ \underset{2}{)}$$

La question c'est (NULL L). Si la réponse est ØUI (ou T) on imprime MAX, puis on va à RE. Dans les LISPs modernes, le CØND tel que je l'ai décrit page 29 se généralise ainsi :

```
(CØND
   ((question1  instruc1  instruc2 .... instrucn ))
   ((question2  instr1  instr2  ...  instrn ))
          :
```

Si une des questions donne une valeur ≠ de NIL, on exécute dans l'ordre les instructions correspondantes instruc1, instruc2, ..., instrucn (de gauche à droite), et on sort du COND en ramenant la valeur de instruction_n. C'est, tel le dauphin, souple, agréable et puissant.

EXERCICE : écrire un programme qui [illegible]

- accepte [illegible]
- effectue la [illegible]
- [illegible]

[illegible]

Ça s'appelle calculer une moyenne.

exemple : la liste (4 [illegible])

la somme [illegible]

4 éléments, je divise 15 par 4 [illegible]

(c'est une division entière).

SOLUTION :

```
(PROG (K N L)
      (SETQ K (SETQ N 0))
      (SETQ L (READ))
RE    (COND
        ((NULL L) (RETURN (QUO K N))))
      (SETQ N (ADD1 N))
      (SETQ K (PLUS K (CAR L)))
      (SETQ L (CDR L))
      (GO RE))
```

L : contient la liste.

N : contiendra le nombre d'éléments de L.

K : contiendra la somme des éléments de L.

Autre façon de faire. Ne cherchez pas trop à comprendre, mais regardez quand même :

```
(DE MOYENNE (L)
    (QUO (APPLY 'PLUS L)
         (LENGTH L)))
```

– ENCØRE [illegible]

Cette fois-ci [illegible]

1 . lire un atome

2 . lire une liste

3 . construire une [illegible]

objets de la liste lue [illegible]

lu en 1 .

exemple : atome : [illegible]

liste : (1 [illegible] 2 [illegible] 3 [illegible])

résultat cherché : (1 2 3 [illegible])

exemple : atome [illegible]

liste : ([illegible])

résultat cherché : ([illegible])

exemple : atome : BØN

liste : (BØN BØN BØN)

résultat cherché : NIL

Bref : éliminer toutes les occurences d'un atome d'une liste

```
1   - - - -            (PRØG  (ØBJ  L  RESU)
2   - - - - -                 (SETQ  ØBJ  (READ))
3   - - - - -                 (SETQ  L  (READ))
4   - - - - -                 (SETQ  RESU  NIL)
5   - - -       RE            (CØND
6   - - - - -                    ((NEQ  (CAR  L)  ØBJ)
7   - - - - -                      (SETQ  RESU  (CØNS  (CAR  L)  RESU))))
8   - - -                     (SETQ  L  (CDR  L))
9   - - - -                   (CØND
10  - - - -                      (L  (GØ  RE)))
11  - - - -                   (RETURN  RESU)   )
```

Tout d'abord, voici l'excellente fonction

(NEQ x y) → { T si x ≠ y ; NIL si x = y }

le contraire de EQ, en somme.

Le problème de ce programme, c'est que la liste résultat, il va falloir la <u>construire</u>. C'est le moment de relire ce que j'ai raconté sur le CONS, pages 4 et 19.

LIGNE 1 : je déclare 3 boîtes ØBJ [ ] L [ ] RESU [ ]

. ØBJ contiendra l'atome lu.
. L contiendra la liste lue.
. RESU contiendra la liste purgée de l'atome.

LIGNES 2 et 3 : j'avale l'atome et la liste.

LIGNE 4 : j'INITIALISE RESU à NIL RESU [NIL].

Pourquoi ? Regardez bien.

exemple : RESU

AVANT [( )]

(SETQ RESU (CØNS 'A RESU))

RESU
APRES [(A)]

(SETQ RESU (CØNS 'B RESU))

RESU
APRES [(B A)]

(SETQ RESU (CØNS 'C RESU))

RESU
APRES [(C B A)]

Donc, de proche en proche, en partant de NIL, on peut <u>construire</u> une liste en rajoutant des éléments en tête. Et de même qu'en page 36, je devais <u>initialiser</u> mon compteur à 0, de même, ici, je dois <u>initialiser</u> RESU à NIL.

LIGNE 5-6-7 : si le CAR de L est différent de l'atome, qui est dans ØBJ, alors cet élément de L est bon à placer dans ma liste-résultat RESU, ce que je fais en ligne 7.

LIGNE 8 : j'avance dans ma liste L.
LIGNE 9-10 : si la liste L est différente de NIL, je vais en RE, où on remet ça.
LIGNE 11 : sinon je ramène ma liste construite RESU.

Il faut remarquer que RESU sera (à l'exception de l'atome), dans l'ordre inverse de L.

<u>exemple :</u> soit

	L1	L2
AVANT	(A B C)	NIL

```
(SETQ L2 (CONS (CAR L1) L2))
(SETQ L1 (CDR L1))
```

	L1	L2
APRES	(B C)	(A)

```
(SETQ L2 (CONS (CAR L1) L2))
(SETQ L1 (CDR L1))
```

	L1	L2
APRES	(C)	(B A)

```
(SETQ L2 (CONS (CAR L1) L2))
(SETQ L1 (CDR L1))
```

	L1	L2
APRES	()	(C B A)

VU ?

les CARs successifs de L1, pris de <u>gauche à droite</u> sont EMPILÉS dans L2 par la tête, de <u>droite à gauche</u>. D'où inversion.

Faites tourner à la main ce programme avec les trois exemples de la page 48, puis passez-le en machine.

Même si ça ne vous dit encore trop rien, jetez un coup d'oeil à cette autre façon de faire :

```
(DE DELQ (OBJ L)
  (COND
    ((NULL L) NIL)
    ((NEQ OBJ (CAR L))
      (CONS (CAR L)
            (DELQ OBJ (CDR L))))
    (T (DELQ OBJ (CDR L))) ))
```

Voir page 90.

- INVERSIØN [illegible]

ET QUELQUES [illegible]

Voici un programme [illegible] une liste dans L, [illegible] renvoie la liste INVERSEE dans RES [illegible]

exemple :
- L = (A B C)
  ↓
  RES = (C B A)
- L = (JE (NE) VOUS AIME (PAS))
  ↓
  RES = ((PAS) AIME VØUS (NE) JE)
- L = ((DØ RE) (MI FA))
  ↓
  RES = ((MI FA) (DØ RE))

```
1.        (PRØG (L RES)
2.             (SETQ L (READ))
3.             (SETQ RES NIL)
4.        RE   (CØND
5.               ((NULL L) (RETURN RES)))
6.             (SETQ RES (CØNS (CAR L) RES))
7.             (SETQ L (CDR L))
8.             (GØ RE) )
```

Etudiez bien ce programme, en conjonction avec ce que je raconte page 50.

Vous voyez la structure de la chose :

ligne 4-5 : je teste si la liste L est vide
si elle ne l'est pas, je vais à la suite, effectue mon petit boulot en ligne 6, puis en
ligne 7 : j'avance dans la liste L, et en
ligne 8 : je recommence le tout.

SCHÉMATIQUEMENT :

```
RE  (CØND
      ((NULL L) passer à autre chose))
         :
      petit boulot
         :
    (SETQ L (CDR L))
    (GØ RE)
```

L'idée c'est : <u>tant que</u> la liste est non-vide, faire un travail, avancer dans la liste !

C'est quelque chose qui revient tellement souvent, qu'une fonction spéciale a été conçue pour cela. Elle a la forme

```
(WHILE    question    instr1  instr2  ...  instrn)
```

Voilà ce qu'elle fait :

. si la "question" ramène une valeur ≠ NIL, on exécute dans cet ordre, les instructions instr1 instr2 ... instrn, puis on recommence le tout (à savoir on repose la question etc.), si d'autre part la "question" ramène NIL, on passe à la suite (où, comme on dit, on sort du WHILE).

AVANTAGES : on évite une étiquette, un CØND, et un GØ.

Une instruction comme celle là, qui REPÈTE, sous condition, une suite d'instructions, on dira que c'est une instruction d'ITÉRATIØN.

exemple : imprimer un par un tous les éléments d'une liste LL :

```
(WHILE  LL     (PRINT  (CAR  LL))
 1             (SETQ  LL  (CDR  LL)) )
                                      1
```

exemple : imprimer 4 fois "BRAVØ"

```
(SETQ  K   4)
(WHILE    (GT  K  0)
 1        (PRINT   'BRAVØ)
          (SETQ  K  (SUB1  K)) )
                               1
```

Muni de cette excellente fonction WHILE, je réécris mon programme d'inversion.

```
(PRØG    (L   RES)
 1       (SETQ   L   (READ))
         (SETQ   RES   NIL)
         (WHILE   L
          2     (SETQ  RES  (CØNS  (CAR  L)  RES))
                (SETQ   L   (CDR   L))  )
                 3               3   2
         (RETURN   RES)  )
                         1
```

Dans ce cas la "question" c'est L, en effet, si L est vide, L = NIL, et on sort du WHILE pour tomber sur le RETURN.

autre exemple : compter dans K le nombre d'éléments d'une liste L

```
(SETQ   K   0)
(WHILE   L
 1           (SETQ  K  (ADD1  K))
             (SETQ  L  (CDR   L))  )
                                   1
```

Le lecteur n'a pas manqué de remarquer, qu'on utilisait TRÈS souvent une suite d'instructions comme :

```
utilisation de (CAR L)
(SETQ L (CDR L))
```

i.e : extraire le 1er élément, puis avancer dans la liste.
On serait fort aise de pouvoir faire les deux d'un coup, le cas est prévu. Voici la fonction :

(NEXTL liste) , elle fait 2 choses

1°/ elle attrape le 1er élément de la liste (comme CAR), ce sera la VALEUR ramenée.
2°/ elle avance dans la liste, ce qui a l'effet d'un (SETQ liste (CDR liste)) implicite.

exemple :

AVANT LL = (A B C)
(PRINT (NEXTL LL)) : imprime A
APRES LL = (B C)
(PRINT (NEXTL LL)) : imprime B
APRES LL = (C)
(PRINT (NEXTL LL)) : imprime C
APRES LL = ()

C'est puissant comme l'éléphant et léger comme la brise.
exemple : imprimer un à un tous les éléments d'une liste L

```
(WHILE L (PRINT (NEXTL L)))
```

C'est le NEXTL qui se charge d'avancer dans la liste.
Je réécris une dernière fois mon programme d'inversion de liste, armé de cette fonction remarquable :

```
(PRØG   (L RES)
        (SETQ L (READ))
        (SETQ RES NIL)
        (WHILE L
           (SETQ RES (CØNS (NEXTL L) RES)) )
        (RETURN RES))
```

Cette fonction NEXTL est plus que commode, elle est également très caractéristique de ce qu'on nomme en programmation un EFFET DE BØRD. En effet, elle ramène bien une valeur, comme toute autre fonction, mais de SURCROÎT, elle MØDIFIE le contenu d'une boîte, ici la variable qui est son argument (revoir page 6).

l'effet de bord, c'est précisément cette modification, ici d'avancer dans la liste-argument.

EXERCICES

[1] a.) que fait cette suite d'instructions ?

```
(SETQ N 7)
(WHILE (GT N 0)
       (PRINT N)
       (SETQ N (SUB1 N)) )
(PRINT 'BØUM)
```

b.) et celle-ci ?

```
(WHILE (SETQ X (READ))
       (PRINT (PLUS X 10)))
```

[2] a.) écrire un programme qui lit en donnée une carte comme celle-ci

```
 7  -1  23  2  ...  18   ()
<------- nombres ------->
```

i.e. une suite de nombres terminée par NIL, puis calcule et imprime la somme de ces nombres.

b.) écrire un programme qui lit une liste exemple : (A B C), puis imprime successivement :

```
(A BRAVØ)
(B BRAVØ)
(C BRAVØ)
```

SØLUTIØNS :

[1] a.) ça imprime :

```
7
6
5
4
3
2
1
BØUM
```

b.) ça lit un nombre (ça vaut mieux), ou bien NIL. Si ce n'est pas NIL, ça imprime la somme de ce nombre et 10, et ça recommence

ex. : lisant

```
3  4  6  -1  ()
```

ça imprime :

```
13
14
16
 9
```

2 a.)

```
(PRØG (X N)
      (SETQ N 0)                    : initialisation !
      (WHILE (SETQ X (READ))
             (SETQ N (PLUS X N)))
      (PRINT N)
      (RETURN 'FINI) )
```

b.)

```
(PRØG (X)
    (WHILE (SETQ X (READ))
           (PRINT (CØNS X '(BRAVØ))) )
    (RETURN 'ØK ))
```

Je vous rappelle que la valeur d'1 (SETQ x y), c'est la valeur de y. En ce sens, la fonction SETQ induit elle aussi un effet de bord : elle
- ramène une valeur, celle de y.
- MØDIFIE la valeur (le contenu) de x.

Je ne fais ici qu'effleurer une notion extrêmement importante, un ENVIRØNNEMENT, c'est un ensemble de VARIABLES (boîtes) et de valeurs associées à ces variables (contenus). En termes crus, il y a effet de bord (en anglais SIDE EFFECT) si, à un moment, il y a modification du contenu d'une ou plusieurs de ces boîtes, donc modification de l'environnement.

- LA FØNCTIØN LIST -

LIST est une fonction à <u>nombre quelconque</u> d'arguments.
(LIST arg1 arg2 ... argn), elle ramène en valeur la liste des valeurs de ses arguments.

exemples :
- (LIST 1 2 3) → (1 2 3)
- (LIST 'A 'B) → (A B)
- (LIST (ADD1 4) (SUB1 3)) → (5 2)

exemples :

si A = FA, B = (LA DØ)

- (LIST A B) → (FA (LA DØ))
- (LIST A) → (FA)
- (LIST A 'A) → (FA A)
- (LIST 2 (CØNS A B) 3) → (2 (FA LA DØ) 3)
- (LIST (NULL T) (NULL NIL)) → (NIL T)
- (LIST 'OUI 'BIEN) → (OUI BIEN)
- (LIST A) → ((LA DØ)

Donc (revoir page 19)

(LIST x1 x2 x3 ... xn) donne un résultat identique à celui que donnerait :

(CØNS x1 (CØNS x2 (CØNS x3 ... (CØNS xn NIL) ...)))

Notez que . (LIST) → NIL .

REVISIØN : Nous commençons à connaître un nombre appréciable de fonctions.

(CAR l) : 1er élément d'une liste l.

(CDR l) : la liste l SANS son 1er élément.

(CØNS x l) : la liste l avec x comme nouveau 1er élément.

(SETQ x y) : place la valeur de y dans la boîte x, et ramène la valeur de y.

(QUØTE x) : x non-évalué.

(PRØG liste de variables ins1 ins2 ... insn) : donne existence aux variables de la liste, et exécute en séquence les instructions ins1, ins2, ..., insn.

La VALEUR d'un PRØG est ou bien . celle de insn (la dernière).
. celle d'un RETURN.

(RETURN x) : sort du PRØG où elle apparaît, en ramenant la valeur de x.

(GØ x) : VERS l'étiquette x. Le problème de la valeur d'un GØ est très intéressant, mais trop complexe pour être traité pour le moment.

(READ) : ramène en valeur un objet lu dans l'univers extérieur.

(PRINT x) : imprime dans l'UE la valeur de x.

(ADD1 x) : x+1

(SUB1 x) : x−1

(PLUS x1 x2 ... xn) : x1 + x2 + ... + xn

(DIFFER x y) : x − y

(TIMES x1 x2 ... xn) : x1 * x2 * ... * xn

(QUØ x y) : x / y

(REM x y) : reste de x / y.

(NULL x) : T si x = NIL, sinon NIL.

(CØND (q1 ins1 ins2 ... insn) (q2 ...) ... (qn ins1 ... insn))

: voir pages 29 et 46.

(LT x y) : T si x < y, sinon NIL.

(GT x y) : T si x > y, sinon NIL.

(EQ x y) : T si l'atome x = l'atome y, sinon NIL.

(ZERØP x) : T si x = 0, sinon NIL.

(WHILE q ins1 ins2 ... insn) : Tant que q ≠ NIL, exécuter ins1, ins2, ... insn; si q = NIL, arrêter et ramener NIL en valeur.

(NEQ x y) : T si atome x ≠ atome y, NIL sinon

(NEXTL l) : ramène (CAR l) et provoque (SETQ l (CDR l))

(LIST x1 x2 ... xn) : (valeur de x1 valeur de x2 ... valeur de xn)